HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

MASTERARBEIT

Herr
José Alberto Vollmann Piña

Clustering variants using an EM approach

2016

Fakultät **Angewandte Computer- und Biowissenschaften**

MASTERARBEIT

Clustering variants using an EM approach

Autor:
José Alberto Vollmann Piña

Studiengang:
Applied Mathematics in Digital Media

Seminargruppe:
MA14w1-M

Erstprüfer:
Prof. Dr. rer. nat. habil. Thomas Villmann

Zweitprüfer:
MSc. David Nebel

Mittweida, 11 2016

Bibliographic information

Vollmann Piña, José Alberto: Clustering variants using an EM approach, 51 Seiten, 24 Abbildungen, Hochschule Mittweida, University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften

Masterarbeit, 2016

Abstract

It is possible to obtain a common updating rule for k-means and Neural Gas algorithms by using a generalized Expectation Maximization method. This result is used to derive two variants of these methods. The use of a similarity measure, specifically the gaussian function, provides another clustering alternative to the before mentioned methods. The main benefit of using the gaussian function is that it inherently looks for a common cluster center for similar data points (depending on the value of the parameter σ). In different experiments we report similar behaviour of batch and proposed variants. Also we show some useful results for the “alternative” similarity method, specifically when there is no clue about the number of clusters in the data sets.

I. Contents

II. List of Figures

III. List of Tables

IV. Preface

The goal of this work is to analyse variants of two of the most used clustering methods nowadays. It has been written to fulfill the graduation requirements of the MSc Applied Mathematics in Digital Media of the University of Applied Sciences Mittweida.

The research topic was suggested by MSc David Nebel together with Prof. Dr. rer. nat. habil. Thomas Villmann. They were always available and willing to help me throughout the whole research and testing period.

I would like to thank my supervisors for their excellent guidance and support during this process. I also wish to thank Mr. Stephan Hartz and my current employer for supporting me and giving me the chance to work in parallel with both projects.

I would like to thank all my mates in the University for their wonderful cooperation as well. It was really nice to share ideas and to study in this program with you. My family deserve also a note of thanks, specially my parents and my two brothers: Cheo, Mima, Uli and Mertio. Last but not least, I would like to thank my dear Laura, who has been supporting me since the beginning of this journey.

Jose Vollmann

Berlin, 25 November 2016.

1 Introduction

The goal of clustering methods is to find groups of similar objects, this is, similar instances are grouped together, while different instances belong to other groups. One of the most known and used methods is k-means algorithm, in which random objects (called frequently prototypes) are moved to the mean of different groups of objects [3].

As k-means is a basic and intuitive, but not robust algorithm, different alternatives have proven to be more efficient [8]. One of these options is the Neural-Gas (NG) method, which is a generalization of the k-means algorithm [6].

The main goal of this work is to adapt a new updating rule to the batch versions of k-means and NG algorithms using the Expectation-Maximization (EM) method. The Expectation-Maximization method maximizes the log-likelihood of a function following an iterative process [7]. It alternates between performing an expectation step (E-step) and a maximization step (M-step). In the first part, a function is created for the expectation of the log-likelihood evaluated using the current estimate for the parameters. Later, parameters are computed maximizing the expected log-likelihood found on the E-step. Finally, calculating these parameters iteratively will result in convergence of the method.

We will show that under the EM approach, we can derive a common updating rule for both proposed algorithms. Furthermore, it will be easier to notice that the Neural Gas is just a generalization of the k-means method, pointing that the main difference is the neighborhood function of the NG method.

An alternative method was also introduced, it uses only the gaussian similarity information between data points and prototypes. The biggest advantage of this algorithm is the automatic determination of number of clusters, however an optimal result depends highly in the choice of the parameter σ of the gaussian function.

The usual batch Neural Gas method is used as a more robust and stable algorithm than usual batch k-means [8], this will be also reported on the results of the proposed alternatives.

2 Theoretical Framework

2.1 Clustering

Cluster analysis, or simply clustering, is the process of grouping a set of objects. The goal is to have items in the same group (called a cluster) that are more similar to each other than to those in other clusters. It is a fundamental problem in machine learning, pattern recognition, image analysis and data compression [9]. Although it is not restricted to those areas, we can find also other applications in computer graphics, finance, among others [11].

Clustering can be carried out by several methods, the difference can be found in the definition of a cluster and how to efficiently find them. Most of the times clusters are defined as groups with small distances among its members, dense areas of the space or intervals or particular statistical distributions. There is not a universal method for this task, the most appropriate clustering algorithm and the choice of parameters will depend on the data set we are working with. The most basic parameter for clustering methods is the number of clusters we want to find, which is usually not easy to know.

In this work we will focus in prototype clustering. The goal of this kind of methods is to distribute prototypes "as good as possible" to represent the data, this is, to set prototypes in data space so they can partition it in different groups containing similar objects. The clusters are defined by the distance of data points to the closest prototype.

A clustering is essentially a set of such clusters. Clusterings can be distinguished as:

- Crisp clustering: Each object belongs to a cluster.
- Fuzzy clustering: Each object belongs to each cluster with certain probability.

In this work we will focus only in crisp clustering, specifically done by prototypes.

Formally, given a set S of data, the clustering structure is represented as a set of k subsets $C = \{C_1, \ldots, C_k\}$ with $C_j \subset S$ for $j = 1, \ldots, k$, such that: $S = \cup_{j=1}^{k} C_j$ and $C_j \cap C_i = \emptyset$ for $j \neq i$. Consequently, any instance in S belongs to exactly one and only one subset C_j.

Many clustering methods use distance measures to determine the dissimilarity between any pair of objects. Throughout this work we will use the euclidean distance, defined for two vectors x_i,$x_j \in \mathbb{R}^d$ as: $||x_i - x_j|| = \sqrt{\sum_{p=1}^{d} (x_i^p - x_j^p)^2}$.

In the next sections (2.1.1 and 2.1.3) we will introduce two known methods used for this task, the k-means and Neural Gas algorithms.

2.1.1 k-means

The term “k-means” was first introduced in 1967 by J.B. MacQueen [3], although the idea was introduced in 1957 by Stuart Lloyd [4]. Despite it is a relative old method, it still remains as the most popular algorithm used to cluster data.

The algorithm aims to partition n objects into k clusters, where the center of each cluster is calculated as the mean of all the instances belonging to that group. This is done by initializing random data-points, called prototypes, which are translated iteratively to the centroid or mean of the closest group of objects. This results in a partitioning of the data space into Voronoi cells [3].

Given a set of observations $(x_1, x_2, \ldots, x_n)$, where each observation is a d-dimensional real vector, the method aims to partition the n observations into $k \leq n$ sets $C = \{C_1, \ldots, C_k\}$ so as to minimize the sum of distance of each point in the cluster to the center. In other words, the objective is to find:

$$\min_{\mathbf{C}} \sum_{j=1}^{k} \sum_{\mathbf{x} \in C_j} \left\| \mathbf{x} - \mathbf{w}_j \right\|^2$$

where w_j is the mean of data points in C_j, for $j = 1, \ldots, k$.

Algorithm

Given an initial set of k prototypes $w_1^{(1)}, \ldots, w_k^{(1)}$, the algorithm proceeds by alternating between two steps:

- Assignment step: Assign each observation to the closest prototype. Geometrically, this means partitioning the data space according to the Voronoi diagram generated by the prototypes.

$$C_j^{(t)} = \left\{ x_p : \left\| x_p - w_j^{(t)} \right\|^2 \leq \left\| x_p - w_i^{(t)} \right\|^2 \; \forall i, 1 \leq i \leq k \right\},$$

  where each x_p is assigned to exactly one C_j ($j = 1, \ldots, k$) in each iteration t.

- Update step: Calculate the new means, in other words, the centroids of the observations in the new clusters.

$$w_j^{(t+1)} = \frac{1}{|C_j^{(t)}|} \sum_{x_i \in C_j^{(t)}} x_i$$

The algorithm stops when the assignments no longer change. Since both steps optimize the objective, and there only exists a finite number of such partitionings, the algorithm must converge to a local optimum. There is no guarantee that the global optimum is found using this algorithm. [5]

In the following graphic we see two iterations of the k-means algorithm, showing how the prototype moves to the centroid of each group:

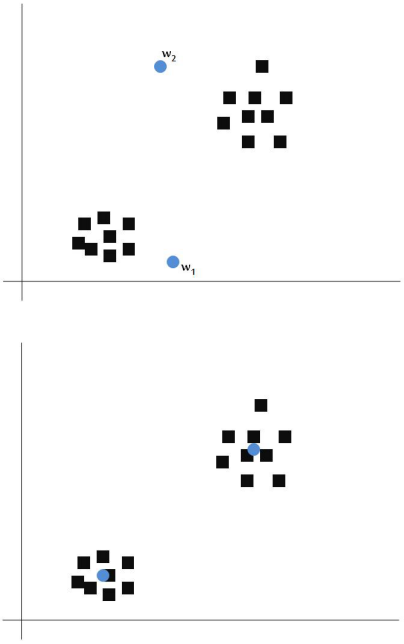


Figure 2.1: Initial and final iterations of k-means

2.1.2 Batch k-means

As stated in [2], we can write the cost function for the batch k-means method as:

$$E_{k-means} = \sum_{j=1}^{k} \sum_{i=1}^{n} X_{I(x_i)}(j) \cdot ||x_i - w_j||^2$$

Where k is the number of prototypes and n is the number of data points.

$X_{I(x_i)}(j)$ is the characteristic function of the winner index $I(x_i)$, which refers to the index of the prototype with minimum distance to x_i (winner). This is, for a given i:

$$I(x_i) = \arg\min_j \{||x_i - w_j||^2 \mid j = 1, \ldots, k\}$$

E is optimized by iteration through the following two adaptation steps until convergence is reached:

- Determine the winner $I(x_i)$ for each data point x_i.
- The new prototype j has to be chosen from the set of data points in the following way:

$$w_j = \sum_{i|I(x_i)=j} \frac{x_i}{|\{i|I(x_i) = j\}|}$$

Disadvantages

Because of its simplicity and intuitiveness, k-means is the most used and known clustering algorithm. However, it has many disadvantages:

1. It gets stuck in local minima, no global optimum solution is guaranteed [5].
2. As we can have different local minima, several runs should be made in order to get a useful result: A solution that approximates the global minimum or just meets our expectations.
3. The algorithm is sensitive to noisy data and outliers (a single outlier can increase the squared error dramatically) [2].
4. It requires the number of clusters in advance, which is not trivial when no prior knowledge is available.
5. Depending on initialization of prototypes, there could be objects that do not belong to any group. They are called “dead units”.

   Remark 2.1 Dead units can be easily avoided when taking initial prototypes from data-points, not at random.

2.1.3 Neural Gas (NG)

Neural gas is an artificial neural network introduced in 1991 by Thomas Martinetz and Klaus Schulten [6]. The neural gas is an algorithm for finding optimal data represen-

tations based on feature vectors. The algorithm is called “neural gas” because of the dynamics of the feature vectors during the adaptation process, which distribute themselves like a gas within the data space [6]. Moreover, it is a generalization of the k-means method and it is used for cluster analysis as a robustly converging alternative. [12]

Algorithm

Assume data points $(x_1, x_2, \ldots, x_n) \in \mathbb{R}^d$ which are distributed according to an underlying distribution P. The goal of NG, as introduced in [6], is to find prototype locations $w_j \in \mathbb{R}^d$, $(j = 1, \ldots, k)$, such that these prototypes represent the distribution P as accurately as possible, minimizing the following cost function:

$$E_{NG}(w) = \frac{1}{2C(\lambda)} \sum_{j=1}^{k} \int h_\lambda(K_j(x,w)) \cdot ||x - w_j||^2 \cdot P(x)dx$$

where $K_{ij} = K_j(x_i, w) = \left|\{w_l \mid ||x_i - w_l||^2 < ||x_i - w_j||^2\}\right|$ is the rank of the prototypes sorted according to the distances.

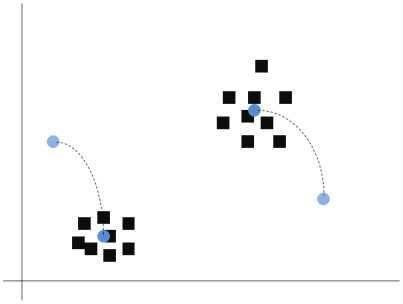
$h_\lambda(t) = \exp(\frac{t}{\lambda})$ is a Gaussian shaped curve with neighborhood range $\lambda > 0$, and $C(\lambda)$ is the constant $\sum_{j=1}^{k} h_\lambda(K_j)$.

The learning rule consists of a stochastic gradient descent, yielding:

$$\Delta w_j = \varepsilon \cdot h_\lambda(k_j(x_i, w)) \cdot (x_i - w_j)$$

for all prototypes w_j given a data point x_i . As pointed in [6], the neighborhood range λ is decreased during training to ensure independence in results for different initializations of w, it also ensures the optimization of the quantization error in the final stages.

Remark 2.2 Unlike k-means, NG is not sensitive to initialization.



2.1.4 Batch NG

It is derived from the cost function of usual NG method, as shown in [2] for discrete data $(x_1, \ldots, x_n)$, reads as:

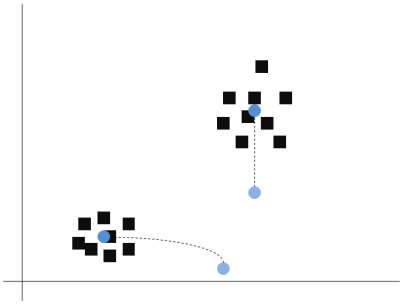


Figure 2.2: Two different prototype initializations yielding the same result

$$E_{NG}(w) = \sum_{j=1}^{k} \sum_{i=1}^{n} h_\lambda \left(K_j(x_i, w) \right) \cdot ||x_i - w_j||^2$$

For the batch algorithm, the quantities $K_{ij} := K_j(x_i, w)$ are treated as hidden variables with the constraint that the values $K_{ij}(j = 1, \ldots, k)$ constitute a permutation of $\{0, \ldots, k-1\}$ for each point x_i .

E_{NG} is interpreted as a function depending on w and K_{ij} which is optimized with respect to the hidden variables K_{ij} and with respect to the prototypes w_j, yielding the two adaptation steps of Batch NG which are iterated until convergence:

1. Determine:

$$K_{ij} = K_j(x_i, w) = \left| \{ w_l \quad | \quad ||x_i - w_l||^2 < ||x_i - w_j||^2 \} \right|$$

   as the rank of prototype w_j.
2. Based on the hidden variables K_{ij} , set:

$$w_j = \frac{\sum_{i=1}^{n} h_\lambda(K_{ij}) \cdot x_i}{\sum_{i=1}^{n} h_\lambda(K_{ij})}$$

It is easy to derive the second step by just taking the derivative of E_{NG} with respect to w_j. A similar calculation will be carried out in section 3.2.

2.2 Kernel and similarity measures

Definition 2.3 A function $k : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ is called a positive definite kernel if it is symmetric, that is, $k(x, x') = k(x', x)$ for any two objects $x, x' \in \mathbb{X}$, and positive definite, that

is:

$$\sum_i \sum_j c_i c_j \cdot k(x_i, x_j) \geq 0$$

for any sequence $(x_1, \ldots, x_n) \in \mathbb{X}$, and any choice of real numbers $c_i \in \mathbb{R}$ for $i = 1, \ldots, n$.

Theorem 2.4 *For any (positive definite) kernel k on a space $\mathbb{X}$, there exists a Hilbert space $\mathbb{F}$ and a mapping $\phi : \mathbb{X} \to \mathbb{F}$ such that*

$$k(x, x') = \langle \phi(x), \phi(x') \rangle$$

for any $x, x' \in \mathbb{X}$ where $\langle \cdot, \cdot \rangle$ represents the inner product in the Hilbert space between any two points.

In the sections 2.1.1 and 2.1.3, we used euclidean distance to measure the dissimilarity between instances. An alternative concept to that of the distance is the similarity function $s(x_i, x_j)$:

A similarity measure or similarity function is a real-valued function that quantifies the similarity between two objects. Although no single definition of a similarity measure exists, usually the measure take on large values for similar objects and either zero or a negative value for very dissimilar objects.

Kernels are often presented as measures of similarity, in the sense that $k(x, x')$ is large when x and $x\prime$ are similar. This motivates the design of kernels for particular types of data or applications, because particular prior knowledge might suggest a relevant measure of similarity in a given context.

The justification for this intuition of kernels as measures of similarity is not always obvious, however we know that kernels are dot products in a feature space (because of 2.4). Yet the notion of dot product does not always fit one's intuition of similarity, which is more related to a notion of distance. There are cases where these notions coincide [16]. Consider, for example, the following kernel on $\mathbb{X} \in \mathbb{R}^d$, called the RBF kernel or Gaussian radial basis function:

$$K_G(x, x') = \exp\big(-\frac{||x - x'||^2}{2\sigma^2}\big)$$

where $\sigma \in \mathbb{R}$ is a free parameter. As proven in [16], this is a kernel which can be written as a dot product $k_G(x, x') = \langle \phi(x), \phi(x') \rangle$. The feature space is a functional space, and an explicit form of the map ϕ is not obvious. As shown in 2.3, we see that this kernel is a decreasing function of the euclidean distance between points, and therefore has a relevant interpretation as a measure of similarity: the larger the kernel $k_G(x, x')$, the closer the points x and x' in $\mathbb{X}$.

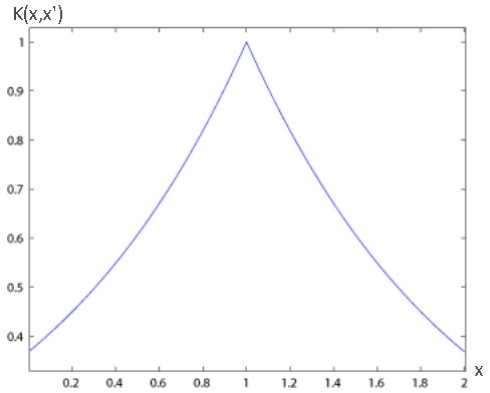


Figure 2.3: Plot of gaussian function for $x \in [0, 2]$ and fixed $x' = 1$, $\sigma = 1$

The goal of this section was to motivate the use of RBF kernel as a similarity measure instead of using the usual euclidean distance for the clustering methods to be proposed later (3.3, 3.4, 4).

2.3 Expectation-Maximization (EM)

The expectation-maximization (EM) algorithm is an iterative method for finding maximum likelihood estimates of parameters in statistical models [7]. The model depends on unobserved latent variables and is used to find (locally) maximum likelihood parameters of a statistical model in cases where the equations cannot be solved directly. The model includes latent variables in addition to unknown parameters and known data observations. In other words, either there are missing values among the data, or the model can be formulated in a simpler way by assuming the existence of additional unobserved data points.

For example, a mixture model can be described more simply by assuming that each observed data point has a corresponding unobserved data point, or latent variable, specifying the mixture component that each data point belongs to. [7]

Finding a maximum likelihood solution typically requires taking the derivatives of the likelihood function with respect to all the unknown values (the parameters and the latent variables) and simultaneously solving the resulting equations.

In general there may be multiple maxima, and there is no guarantee that the global maximum will be found.

2.3.1 General algorithm

Given the statistical model which generates a set $\mathbb{X}$ of observed data, a set of, possible unobserved, latent data or missing values $\mathbb{Z}$, and a vector of unknown parameters θ, along with a likelihood function $L(\theta;\mathbb{X},\mathbb{Z}) = p(\mathbb{X},\mathbb{Z}|\theta)$, the maximum likelihood estimate (MLE) of the unknown parameters is determined by the marginal likelihood of the observed data:

$$L(\theta;\mathbb{X}) = p(\mathbb{X}|\theta) = \sum_{\mathbb{Z}} p(\mathbb{X},\mathbb{Z}|\theta)$$

However, this quantity is often intractable (e.g. if $\mathbb{Z}$ is a sequence of events, so that the number of values grows exponentially with the sequence length, making the exact calculation of the sum extremely difficult) [14].

The EM algorithm seeks to find the MLE of the marginal likelihood by iteratively applying the following two steps:

- Expectation step (E step): Calculate the expected value of the log likelihood function, with respect to the conditional distribution of $\mathbb{Z}$ given $\mathbb{X}$ under the current estimate of the parameters $\theta^{(t)}$:

$$Q(\theta|\theta^{(t)}) = \mathrm{E}_{\mathbb{Z}|\mathbb{X},\theta^{(t)}}\left[\log L(\theta;\mathbb{X},\mathbb{Z})\right]$$

- Maximization step (M step): Find the parameter that maximizes this quantity:

$$\theta^{(t+1)} = \arg\max_{\theta} Q(\theta|\theta^{(t)})$$

Remark 2.5 In the k-means and in the NG algorithms, the “assignment step” and the “update step” can be seen as **expectation step** and **maximization step** respectively.

3 New updating formula

In this chapter we will show how the EM method is used to derive a new updating rule for the k-means and the NG algorithms.

3.1 The generalized EM approach for clustering

As pointed out in the previous chapter (Section 2.3.1), the gEM makes use of hidden variables z_k such that the following is valid:

$$P(x_i \mid w_j) = \sum_k P(z_k)P(x_i \mid z_k)$$

In this sense, the function $L(X,w)$ should be maximized using an iterative alternating EM-scheme. As shown previously, during the expectation step, the distribution $\hat{P} = P(z_k \mid x_i, w_j)$ for given w_j is determined. And later, in the maximization step, the prototypes w_j are adapted such that the expected value: $E_{\hat{P}}[\log(P(x_i, z_k \mid w_j))]$ is improved.

Moreover, the generalized EM is an optimization scheme developed for likelihood functions of the form:

$$L(X,w) = \sum_i \log \sum_j P(x_i \mid w_j)$$

where $P(x_i \mid w_j)$ is the joint probability that the data point x_i is generated by the prototype w_j.

Following the process described in [1], an alternative scheme of the EM algorithm is introduced:

We assume a logarithmic function such that:

$$L(X,w) = \sum_i \log \sum_j g(x_i, w_j)$$

with positive, bounded real functions g. In this case the g are not longer assumed to be

probabilities as before.

Using the general functions $g(x_i, w_j)$, we define a formal probability:

$$p(w_j \mid x_i) = \frac{g(x_i, w_j)}{\sum_j g(x_i, w_j)}$$

for a prototype w_j given x_i.

We assume also arbitrary real numbers $\gamma_{ij} = \gamma(w_j \mid x_i)$ fulfilling the restriction: $\gamma_{ij} \geq 0$ and $\sum_{j=1}^{M} \gamma_{ij} = 1$, for all $i = 1, \ldots, N$ which can be interpreted as formal probability values.

Now we define the formal Kullback-Leibler divergence (KLD) [13]:

$$\kappa(\gamma \mid p) = \sum_i \sum_j \gamma_{ij} \log \Big(\frac{\gamma_{ij}}{p(w_j \mid x_i)} \Big)$$

being always non-negative and such that the following is valid:

$$\kappa(\gamma \mid p) = 0 \Leftrightarrow \gamma_{ij} = p(w_j \mid x_i) \forall i, j$$

Remark 3.1 The KLD is defined as a measure of the difference between two given probability distributions P and Q. Where P typically represents the "true" distribution of data, observations, while Q typically represents a theory, model, description, or approximation of P. Taking $\kappa(\gamma \mid p) = 0$ means that no information is lost when $\gamma = p$.

We will show that the likelihood function can be decomposed as: $L(X, w) = \iota(\gamma, w) + \kappa(\gamma \mid p)$, where ι is a loss term defined as:

$$\iota(\gamma, w) = \sum_i \sum_j \gamma_{ij} \log \frac{g(x_i, w_j)}{\gamma_{ij}}$$

.

In order to prove the decomposition, we follow the steps from [1], i.e. the proof for the approach for median LVQ variants, however we do not involve data labels:

Proof:

$$
\begin{aligned}
L(X,w) &= \sum_{i=1}^{n} \log\big(\sum_{j=1}^{k} g(x_i,w_j)\big) \\
&= -\sum_{i=1}^{n}\big(\sum_{j=1}^{k}\gamma(w_j \mid x_i)\big)\log\Big(\frac{1}{\sum_{l=1}^{k} g(x_i,w_l)}\Big) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{k}\gamma(w_j \mid x_i)\log\Big(\frac{g(x_i,w_j)}{\gamma(w_j \mid x_i)}\Big) - \sum_{i=1}^{n}\sum_{j=1}^{k}\gamma(w_j \mid x_i)\log\Big(\frac{g(x_i,w_j)}{\gamma(w_j \mid x_i)}\Big) - \\
&- \sum_{i=1}^{n}\sum_{j=1}^{k}\gamma(w_j \mid x_i)\log\Big(\frac{1}{\sum_{l=1}^{k} g(x_i,w_l)}\Big) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{k}\gamma(w_j \mid x_i)\log\Big(\frac{g(x_i,w_j)}{\gamma(w_j \mid x_i)}\Big) - \sum_{i=1}^{n}\sum_{j=1}^{k}\gamma(w_j \mid x_i)\log\Big(\frac{g(x_i,w_j)}{\sum_{l=1}^{k} g(x_i,w_l)\gamma(w_j \mid x_i)}\Big) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{k}\gamma(w_j \mid x_i)\log\Big(\frac{g(x_i,w_j)}{\gamma(w_j \mid x_i)}\Big) - \sum_{i=1}^{n}\sum_{j=1}^{k}\gamma(w_j \mid x_i)\log\Big(\frac{p(w_j \mid x_i)}{\gamma(w_j \mid x_i)}\Big) \\
&= \sum_{i=1}^{n}\iota_i(\gamma,w) + \sum_{i=1}^{n}\kappa_i(\gamma \mid p) \\
&= \iota(\gamma,w) + \kappa(\gamma \mid p)
\end{aligned}
$$

□

Finally, the usual EM steps are:

- E-Step: Due to the fact that $\kappa(\gamma \mid p) = 0$ is valid, we set $\gamma = p$, and we have that:

$$L(X,w) = \iota(\gamma,w) = \sum_i \sum_j \gamma_{ij}\log\Big(\frac{g(x_i,w_j)}{\gamma_{ij}}\Big)$$

- M-Step: We look for a w that maximizes ι. This will be derived in next section 3.2.

3.2 Derivation of the updating rule

From previous section 3.1, we have that our likelihood function can be written as:

$$L(\gamma,w) = \sum_i \sum_j \gamma_{ij}\log\Big(\frac{g(x_i,w_j)}{\gamma_{ij}}\Big)$$

Now we want to look for a w that maximizes this function, this means that we want to solve the following optimization problem:

$$\frac{\partial}{\partial w_j} L(\gamma, w) = \frac{\partial}{\partial w_j} \sum_i \sum_j \gamma_{ij} \log\Big(\frac{g(x_i, w_j)}{\gamma_{ij}}\Big) = 0$$

Instead of using the euclidian distance, we want to use a kernel or a similarity measure. As motivated in section 2.2, we will use the gaussian kernel.

First we will derive the simplest case:

$$g = s(x_i, w_j) = \exp\Big(-\frac{||x_i - w_j||^2}{2\sigma^2}\Big)$$

Then we get:

$$\begin{aligned}
0 &= \frac{\partial}{\partial w_j} \sum_i \sum_j \gamma_{ij} \log\Big(\frac{s(x_i, w_j)}{\gamma_{ij}}\Big) \\
&= \frac{\partial}{\partial w_j} \sum_i \sum_j \gamma_{ij} \log\Big(\exp\Big(-\frac{||x_i - w_j||_2^{2\,2}}{2\sigma}\Big)\Big) \\
&= \frac{\partial}{\partial w_j} \sum_i \sum_j \gamma_{ij} \Big(-\frac{||x_i - w_j||_2^2}{2\sigma^2}\Big) \\
&= -\frac{1}{2\sigma^2} \frac{\partial}{\partial w_j} \sum_i \sum_j \gamma_{ij} (||x_i - w_j||_2^2) \\
&= -\frac{1}{2\sigma^2} \sum_i \frac{\partial}{\partial w_j} \sum_j \gamma_{ij} (||x_i - w_j||_2^2) \\
&= -\frac{1}{2\sigma^2} \sum_i \gamma_{ij} 2(||x_i - w_j||_2) \frac{\partial}{\partial w_j} (||x_i - w_j||_2) \\
&= -\frac{2}{2\sigma^2} \sum_i \gamma_{ij} (\sqrt{(x_i - w_j)^2}) \frac{\partial}{\partial w_j} (\sqrt{(x_i - w_j)^2}) \\
&= -\frac{1}{\sigma^2} \sum_i \gamma_{ij} (x_i - w_j) \frac{\partial}{\partial w_j} (x_i - w_j) \\
&= -\frac{1}{\sigma^2} \sum_i \gamma_{ij} (x_i - w_j)(-1) \\
&= \frac{1}{\sigma^2} \sum_i \gamma_{ij} (x_i - w_j)
\end{aligned}$$

We can generalize this result for both methods using a common cost function (as introduced in [2]):

$$E = \sum_i \sum_j r_{ij} s(x_i, w_j)$$

where s is a similarity measure and r is is the characteristic function of the winner, $X_{I(x_j)}(i)$ for k-means, and it is the neighborhood function for neural gas $h_\lambda(k_{ij}(w)) = \exp(-\frac{K(x_i,w_j)}{2\lambda})$, where K is the ranking function and $\lambda \in \mathbb{R}^+$ is the neighborhood range.

We now solve the more general problem:

$$\frac{\partial}{\partial w_j} \sum_i \sum_j \gamma_{ij} \log\Big(\frac{r_{ij} s(x_i, w_j)}{\gamma_{ij}}\Big) = 0$$

This is:

$$\begin{aligned}
0 &= \frac{\partial}{\partial w_j} \sum_i \sum_j \gamma_{ij} \log\Big(\frac{r_{ij} s(x_i, w_j)}{\gamma_{ij}}\Big) \\
&= \frac{\partial}{\partial w_j} \sum_i \sum_j \gamma_{ij} \big(\log(r_{ij}) + \log(s(x_i, w_j))\big) \\
&= \frac{\partial}{\partial w_j} \sum_i \sum_j \gamma_{ij} \big(\log(r_{ij})\big) + \frac{\partial}{\partial w_j} \sum_i \sum_j \gamma_{ij} \big(\log(s(x_i, w_j))\big) \\
&= \big(0\big) + \frac{\partial}{\partial w_j} \sum_i \sum_j \gamma_{ij} \big(\log(s(x_i, w_j))\big) \\
&= \frac{1}{\sigma} \sum_i \gamma_{ij} (x_i - w_j)
\end{aligned}$$

Rearranging the previous equation, we obtain a closed solution for w_j:

$$\begin{aligned}
0 &= \frac{1}{\sigma^2} \sum_i \gamma_{ij} (x_i - w_j) \\
\Rightarrow 0 &= \sum_i \gamma_{ij} x_i - \sum_i \gamma_{ij} w_j \\
\Rightarrow \sum_i \gamma_{ij} x_i &= \sum_i \gamma_{ij} w_j \\
\Rightarrow w_j &= \frac{\sum_i \gamma_{ij} x_i}{\sum_i \gamma_{ij}}
\end{aligned}$$

Where:

$$\gamma_{ij} = \frac{g(w_j, x_i)}{\sum_i \sum_j g(w_j, x_i)}$$

This means that we have a common updating rule for both algorithms, however the difference lies in the computation of γ. Note that γ is a function of g, which in turn depends on z.

3.3 Proposed k-means algorithm

Using our last result from section 3.2, we can reformulate the k-means algorithm in the following way:

1. Determine the winner for each data point x_j.
2. Compute γ (from EM method) using the winner information:

$$\gamma_{ij} = \frac{r_{ij} s(w_j, x_i)}{\sum_i \sum_j r_{ij} s(w_j, x_i)}$$

   where $r_{ij} = X_{I(x_j)}(i)$ and $s(w_j, x_i) = \exp(-\frac{||x_i - w_j||^2}{2\sigma^2})$
3. Compute/Update the prototypes using the formula:

$$w_j = \frac{\sum_i \gamma_{ij} x_i}{\sum_i \gamma_{ij}}$$

4. Repeat until convergence

3.4 Proposed NG algorithm

In a similar way as in k-means algorithm, we can reformulate NG as:

1. Determine ranks of prototypes using γ from EM method:

$$K_{ij} = K_j(x_i, w) = \left| \{ w_l | s(x_i, w_l) > s(x_i, w_j) \} \right|$$

   where $s(x_i, w_j) = \exp(-\frac{||x_i - w_j||^2}{2\sigma^2})$

2. Compute γ (from EM method) using the ranking information:

$$\gamma_{ij} = \frac{r_{ij}s(w_j, x_i)}{\sum_i \sum_j r_{ij}s(w_j, x_i)}$$

where $r_{ij} = \exp(-\frac{K(x_i, w_j)}{2\lambda})$ and $s(w_j, x_i) = \exp(-\frac{||x_i - w_j||^2}{2\sigma^2})$

3. Update the prototypes using the common updating rule:

$$w_j = \frac{\sum_i \gamma_{ij} x_i}{\sum_i \gamma_{ij}}$$

4. Repeat until convergence

4 Alternative method

In section 3.2, the generalized EM method for clustering was introduced, in which the likelihood function was written as $L(X,w) = \sum_i \log \sum_j g(x_i, w_j)$ with positive, bounded real functions g.

Note that $g = s(x_i, w_j)$ meet these properties, so we can introduce another alternative to the proposed k-means and NG methods by just taking similarity information, in other words, we do not take into account the Winner/Neighborhood function. This would be equivalent to minimizing the following cost function:

$$E = \sum_i \sum_j s(x_i, w_j)$$

The trivial solution would be finding the center or mean of the whole data set. However, as we are using the RBF kernel, depending on the parameter σ we will look for the means of the closest group to each prototype. In other words, the σ parameter is taken as a range parameter: For a big γ value, we will get fewer means (or just the center of the data set); on the contrary, for an small σ value, we will get more means of different groups.

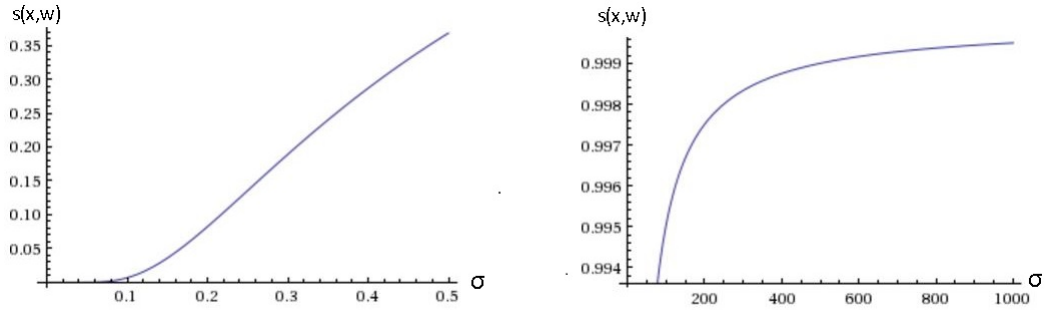


Figure 4.1: σ vs. Similarity measure (For fixed $||x-w||^2 = 0.5$)

It was shown in 3.2 that the updating formula for this case is the same as the one for proposed k-means and NG methods, however in this case it is not necessary to compute “winners”' or “rankings”, just similarities.

The big disadvantage of this method is that we rely too much on the choice of the gaussian similarity parameter, however we can use it as an alternative when there is no clue about the number of clusters in the data set.

Other disadvantage is that we have to initialize a big quantity of prototypes, ideally uniformly distributed in the data space. If not, the method is not likely to find an optimal

solution.

In next graphics we show an easy geometrical interpretation of how the σ value influence the final result of a 2-cluster random generated data:

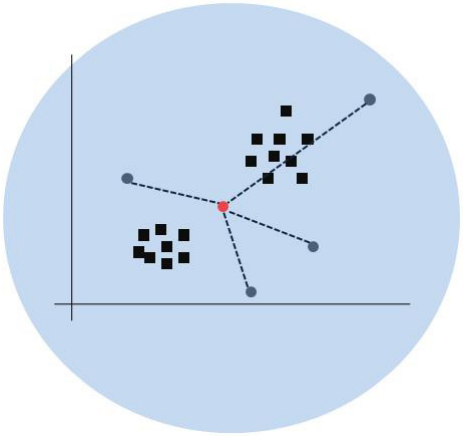


Figure 4.2: Big value of σ: All data-points are similar

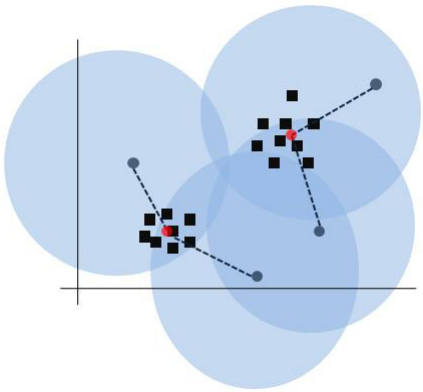


Figure 4.3: Small enough value of σ: Neighbor data points are similar

4.1 Algorithm

1. Initialize enough prototypes (Uniformly distributed in data space)
2. Compute γ (from EM method):

$$\gamma_{ij} = \frac{g(w_j, x_i)}{\sum_i \sum_j g(w_j, x_i)}$$

where $g(w_j, x_i) = s(x_i, w_j) = \exp\left(-\frac{||x_i - w_j||^2}{2\sigma^2}\right)$

3. Compute/Update the prototypes using the formula:

$$w_j = \frac{\sum_i \gamma_{ij} x_i}{\sum_i \gamma_{ij}}$$

4. Repeat until convergence

5 Testing data-sets

We compared usual batch and the proposed methods in five different data sets:

5.1 Four-cluster random data

Four groups of data, each one of them containing ten data points, are build as follows:

```
four_cluster =
[ rand(10,2);
  rand(10,2) + 2;
  rand(10,2) + 4;
  rand(10,2) + 6 ];
```

So we have a 40×2 data set.

The following graphic is an example of it:

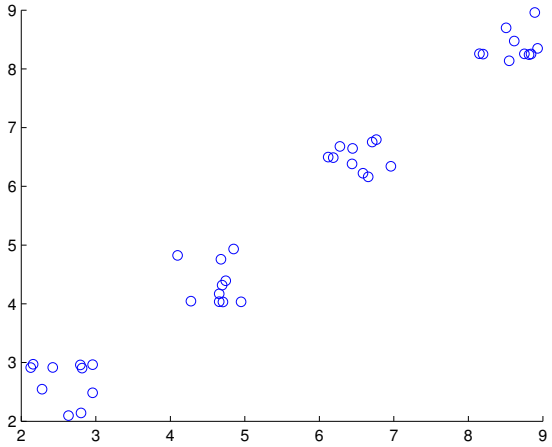


Figure 5.1: Four-cluster data set

5.2 Matlab's "clusterdemo" data

It consists of a 600×3 array, containing three clusters:

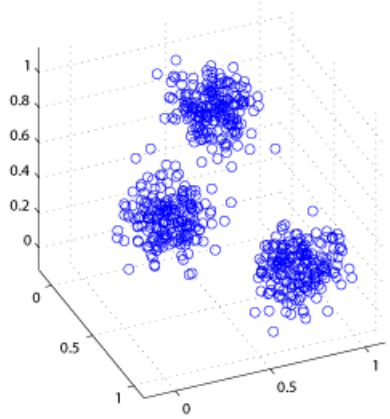


Figure 5.2: “clusterdemo” data set

5.3 Mouse data

978×2 data set whose scatter plot is similar to cartoon Mouse shape.

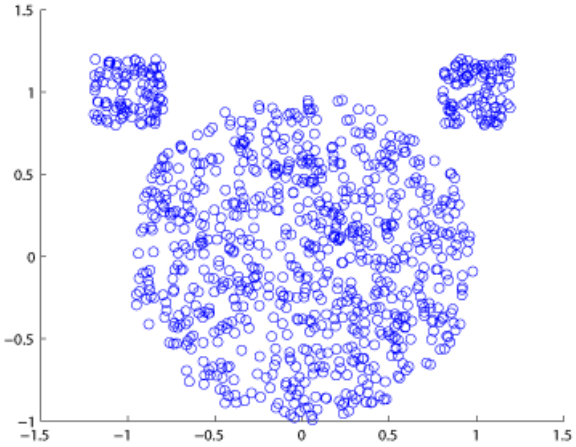


Figure 5.3: Mouse data set

5.4 CI data

It is a 1000×2 data set. It consists of data points whose scatter plot show the letter ’C’ and the letter ’I’.

Below is the graphic of the data set:

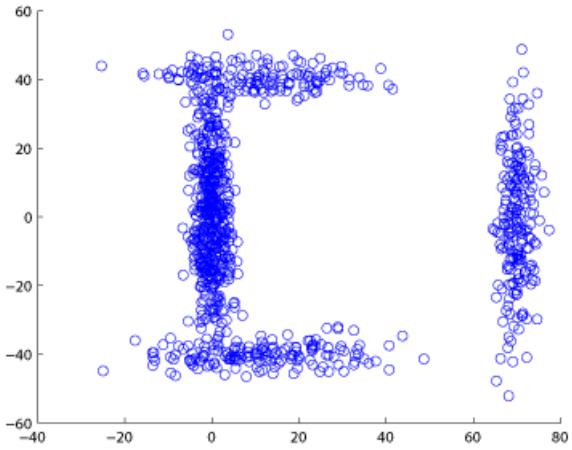


Figure 5.4: CI data set

5.5 High/Low density data

Random data set with six different clusters: Three groups with high density of data points, each having 5000×2 elements; and three groups with low density, each containing 50×2 elements. This means we have a data set of size 15150×2.

Data was generated in a similar way as in the “Four cluster random” data set:

```
density_data =
[(1100-900).*rand(5000,2) + 900;
(2000-1800).*rand(5000,2) + 1800;
[(2000-1800).*rand(5000,1) + 1800, (1100-900).*rand(5000,1) + 900];
[(3300-2800).*rand(50,1) + 2800, (1550-1450).*rand(50,1) + 1450];
(1600-1400).*rand(50,2) + 1400;
[(1100-900).*rand(50,1) + 900, (2100-1900).*rand(50,1) + 1900]];
```

The following graphic shows the “density” data set:

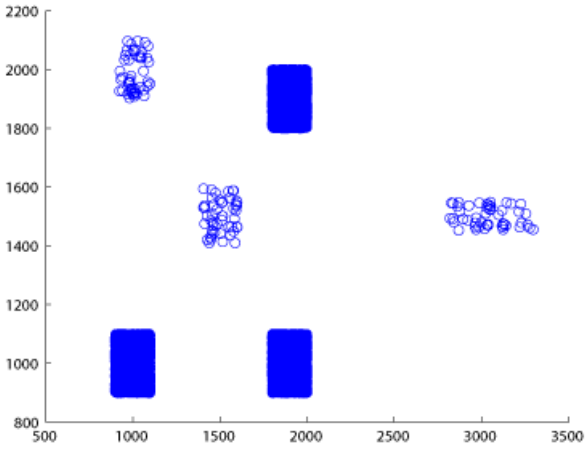


Figure 5.5: Density data set

6 Experiments and results

All experiments were run in a portable PC equipped with Windows Vista 32 Bits, Intel dual-core CPU 2.26GHz and 4GB RAM. The algorithms were coded in Matlab (version R2014a).

We run fifty times each algorithm and then took different statistics from the samples. For all cases we used the post-labeling accuracy measure, this is we labeled the data after clustering according to the the closest prototype, then we compare results with the "original" labels that were set.

The required tolerance to find the optimum was fixed in all methods to 10^{-5}, this means that the algorithm runs until the difference in distance between the current prototype and the previous is smaller than this number.

For all k-means and NG methods, we initialized the prototypes as points from the data set taken at random:

```
Input data-set X
init w
r = Random permutation of indices of the data set
for i = 1 to Number of prototypes
    w(i,:) = X(r(i),:);
endfor
```

For the "alternative method" we initialized a grid of 121 prototypes as follows (for the case of "clusterdemo" we used 125 prototypes, having a $5 \times 5 \times 5$ grid):

```
Input data-set X= %2-dimensional
init w
xaxis = min(x) : (max(x)-min(x))/10 : max(x);
yaxis = min(y) : (max(y)-min(y))/10 : max(y);
for i = 1 to length(xaxis)
    for j = 1 to length(yaxis)
        w=[w(:,1) , w(:,2); xaxis(i), yaxis(j)];
    endfor
endfor
```

For the case of Neural Gas methods, we used the decrease λ formula used in [6], this is:

$$\lambda = \lambda_{initial}\left(\frac{\lambda_{final}}{\lambda_{initial}}\right)^{\frac{it}{maxit/10}}$$

Where $\lambda_{initial} = 1$, $\lambda_{final} = 0.0001$, it is the current iteration and $maxit$ is the maximum number of allowed iterations, in our case 1000.

6.1 Data set I: Four-cluster random data

6.1.1 4 prototypes

Method	σ	Min Iterations	Max Iterations	Mean Iterations	Accuracy	CPU Time (sec)
Batch k-means	NA	2	5	3.18	60%	0.01
Batch NG	NA	30	30	30	100%	0.55
Proposed k-means	100	3	7	4.62	48%	0.03
Proposed NG	100	30	30	30	100%	0.56
Alternative	1	19	19	19	100%	0.80

Table 6.1: Results for “four-cluster” data set (using 4 prototypes)

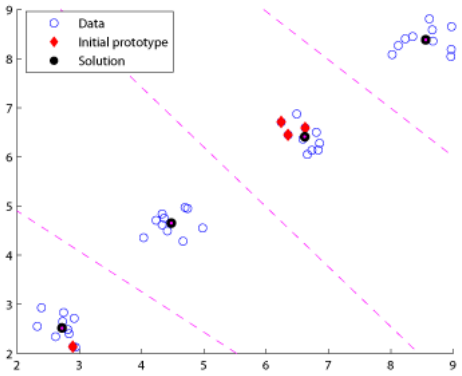


Figure 6.1: Expected result for “four-cluster” data set

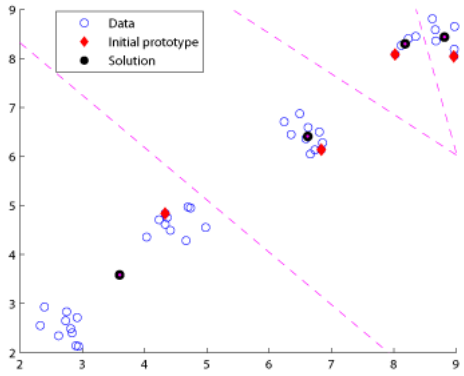


Figure 6.2: A k-means not optimal clustering for “four-cluster” data set

6.2 Data set II: “clusterdemo”

6.2.1 3 prototypes

Method	σ	Min Iterations	Max Iterations	Mean Iterations	Accuracy	CPU Time (sec)
Batch k-means	NA	5	45	15.18	88%	0.19
Batch NG	NA	29	29	29	100%	0.64
Proposed k-means	100	2	14	6.36	92%	0.13
Proposed NG	100	29	29	29	100%	0.62
Alternative	0.01	24	24	24	100%	8.02

Table 6.2: Results for “clusterdemo” data set (using 3 prototypes)

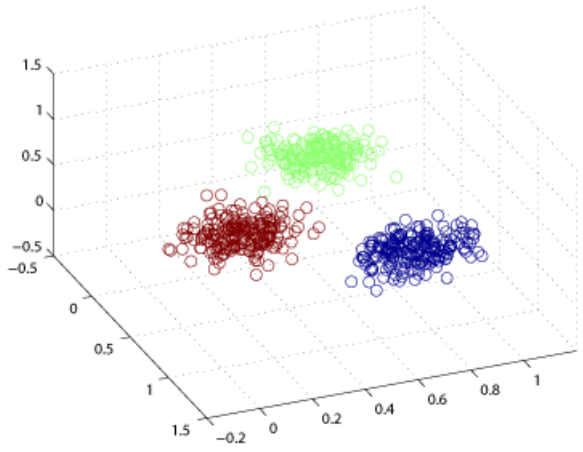


Figure 6.3: Expected clustering for “clusterdemo” data set

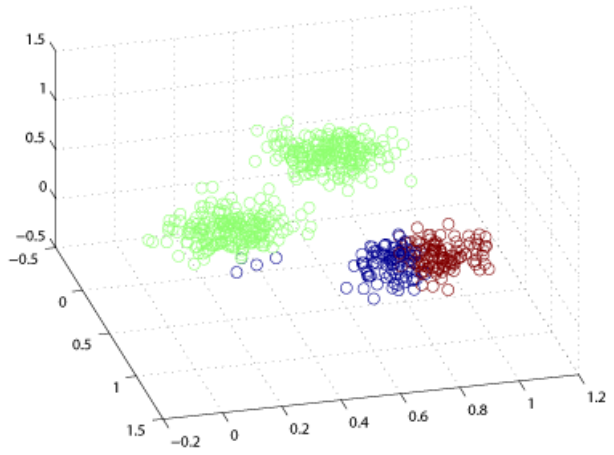


Figure 6.4: Not optimal clustering found with k-means “clusterdemo” data set

6.3 Data set III: Mouse data

6.3.1 3 prototypes

Method	σ	Min Iterations	Max Iterations	Mean Iterations	Accuracy	CPU Time (sec)
Batch k-means	NA	4	20	9.96	79%	0.04
Batch NG	NA	29	29	29	79%	1.18
Proposed k-means	1,000	7	17	13.66	80%	0.12
Proposed NG	1,000	29	29	29	79%	1.15
Alternative	0.15	124	124	124	83%	40.23

Table 6.3: Results for mouse data set (using 3 prototypes)

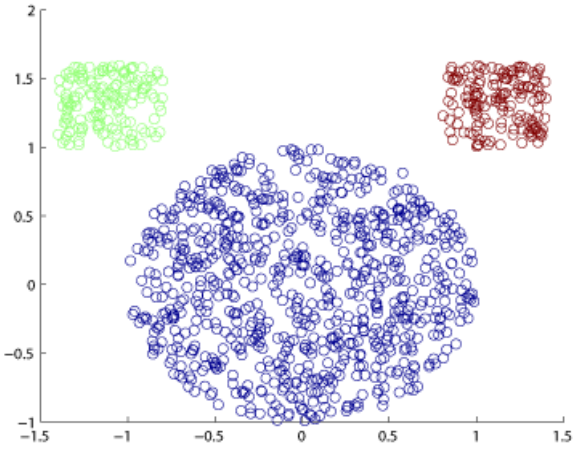


Figure 6.5: Expected clustering of Mouse data

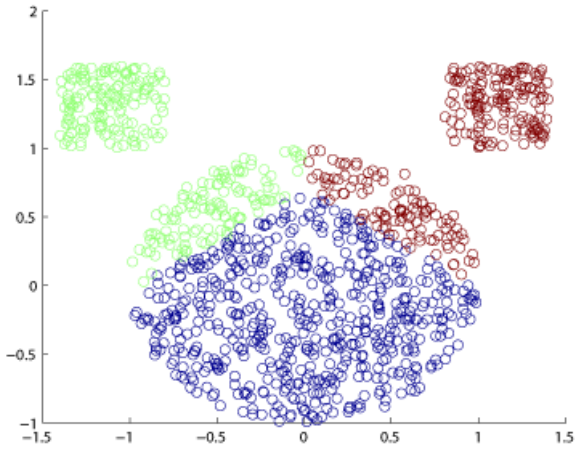


Figure 6.6: Three different clusters colored according to prototype distances ($\approx 20\%$ error)

6.4 Data set IV: CI data

We will show two different results for the CI data set:

- Expecting 2 clusters: Letter C and letter I.
- Expecting 4 clusters: Each part of the letter C (upper, vertical and lower lines of the letter C) and the complete letter I.

6.4.1 2 prototypes

Method	σ	Min Iterations	Max Iterations	Mean Iterations	Accuracy	CPU Time (sec)
Batch k-means	NA	4	25	15.88	99%	0.15
Batch NG	NA	33	33	33	99%	1.12
Proposed k-means	10,000	5	21	13.66	89%	0.21
Proposed NG	10,000	33	33	33	99%	1.01
Alternative	100	82	82	82	99%	23.77

Table 6.4: Results for CI data (2 clusters)

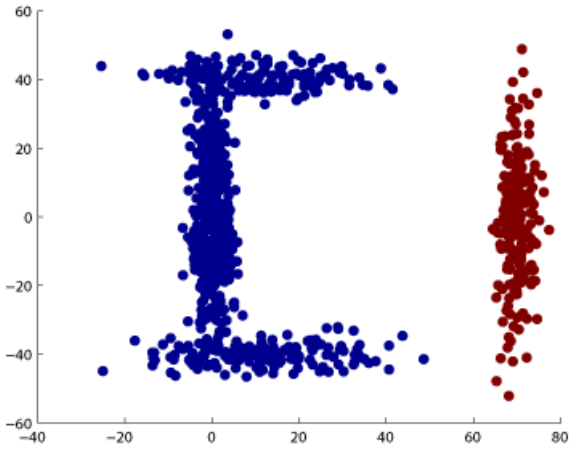


Figure 6.7: Expected 2 cluster result for CI data

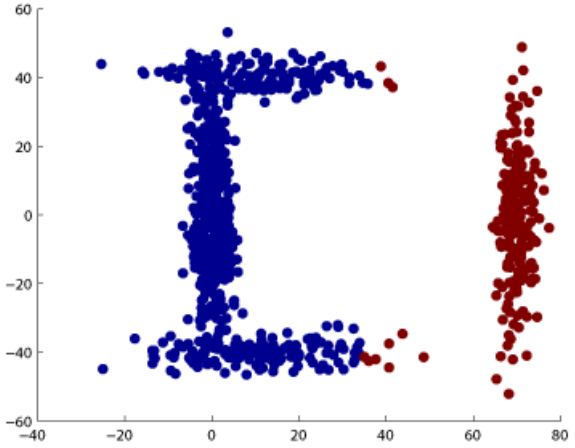


Figure 6.8: 2 clusters obtained for CI data (error around 1% and 10%)

6.4.2 4 prototypes

Method	σ	Min Iterations	Max Iterations	Mean Iterations	Accuracy	CPU Time (sec)
Batch k-means	NA	4	23	12.42	50%	0.16
Batch NG	NA	32	32	32	89%	1.31
Proposed k-means	$10,000$	8	39	18.78	79.70%	0.33
Proposed NG	$10,000$	32	32	32	89%	1.32
Alternative	50	123	123	123	89%	33.82

Table 6.5: Results for CI data (4 clusters)

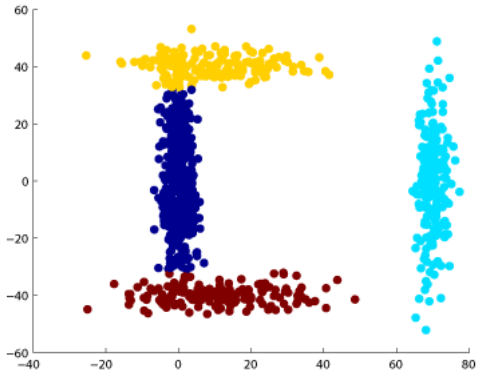


Figure 6.9: Expected 4 cluster result for CI data

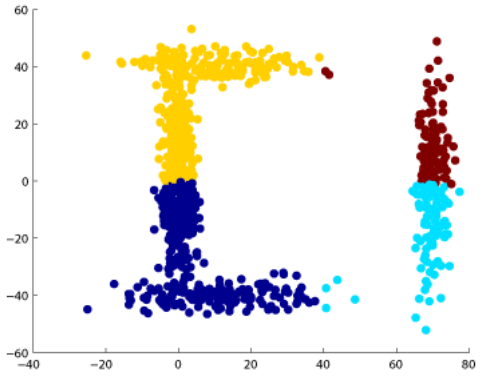


Figure 6.10: 4 clusters obtained for CI data using k-mean methods (error around 50%)

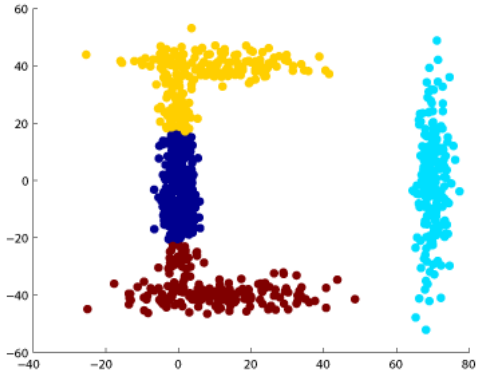


Figure 6.11: 4 clusters obtained for CI data using NG and alternative methods (error around 10%)

6.5 Data set V: Density data

6.5.1 6 prototypes

The goal of this experiment is to show how data density affect the clustering results:

Method	σ	Min Iterations	Max Iterations	Mean Iterations	Accuracy	CPU Time (sec)
Batch k-means	NA	17	24	20.96	59.33%	5.56
Batch NG	NA	33	42	35.78	53.11%	38.47
Proposed k-means	1,000,000	17	48	33.15	72.55%	13.12
Proposed NG	1,000,000	31	43	36.13	52.51%	37.14
Alternative	10,000	41	41	41	100%	126.48

Table 6.6: Results for density data set (using 6 prototypes)

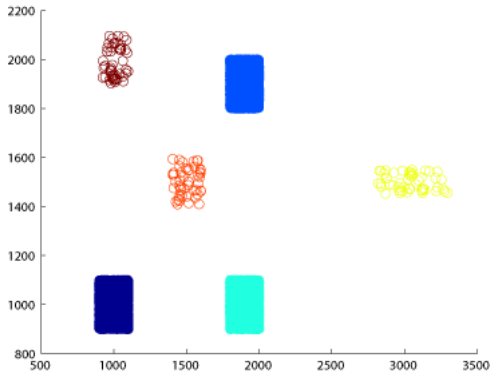


Figure 6.12: Expected clustering of density data

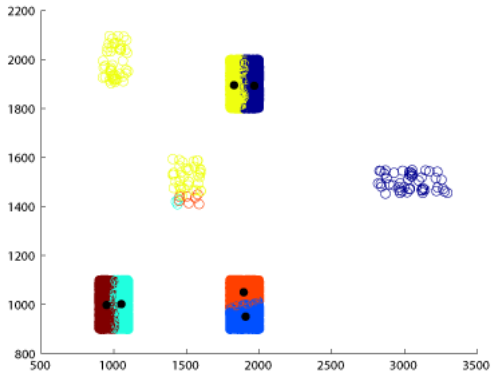


Figure 6.13: Clusters colored according to prototype distances ($\approx 50\%$ error)

6.6 Analysis of results

As expected, the usual batch and proposed Neural Gas methods had more stable and accurate results than k-means counterpart in all the cases we tested.

We noticed that the proposed k-means variant is still very sensitive to prototype initialization. This behaviour was expected due to the fact that we are still using the same winner approach as in regular method.

The proposed k-means algorithm reported slightly better accuracy than the batch variant. However, in most of the cases the proposed method converged to a solution in 150% more time than the usual batch method.

In all the test cases, the proposed NG variant had similar behaviour than the batch method. Moreover, both methods converged to the same optima in equal number of iterations.

Remark 6.1 For the NG algorithms we have to store the K or γ matrices (for batch and proposed methods respectively). This could lead to a memory problem for big data sets, note that $K, \gamma \in \mathbb{R}^{n \times k}$, where n is the number of data-points and k is the number of prototypes).

The "alternative" method had similar results to the NG variants, however the computational time was higher in every test case. This was expected due to the fact that the number of prototypes was fixed to 121. In most of the cases, a grid with fewer prototypes would have also reached the same results, however the goal was to pick a unique scenario in order to compare methods with the same configuration.

In the last test case (density data set), the alternative method outperformed all other algorithms. As reported in [10], the k-means method tends to translate prototypes to the groups with more data density, such data is known in literature as "imbalanced data" [10]. Our results show that the "alternative" method can be used in such cases whether we know or not the number of clusters in advance.

7 Conclusions

We could derive a common updating rule for k-means and Neural Gas methods using an Expectation-Maximization approach. In this way, it was easy to verify that NG is a generalization of k-means where the main difference is the definition of Winner/Neighborhood function.

An alternative method was introduced, in which just the (gaussian) similarities between prototypes and data points are taken into account. We noticed that the ability to find an optimum on this method depends on the choice of the σ parameter, for which we showed an intuitive geometrical interpretation.

Five different data sets were tested in order to have better understanding of the methods and to compare the performance in different situations.

We noticed that proposed methods performed very similar to the batch versions. According to the results, the proposed k-means reported better accuracy than its batch counterpart, taking approximately the double of time than usual batch variant. For the case of Neural Gas methods, the proposed NG method can be a plausible option in case similarity measure is preferred over usual euclidean distance.

There is evidence that the proposed k-means method is still sensible to prototype initialization, which was not noticed in the Neural Gas methods. This is still the biggest advantage of NG over k-means.

The disadvantage in NG methods is the calculation of the K and γ matrices, for batch and proposed variants respectively, which in large data sets could slow down drastically the time to find an optimum and could lead to memory problems. The size of the matrix is $n \times k$ (being n the number of data points and k the number of prototypes).

In the case of the alternative method, the biggest advantage over all other variants is the automatic determination of the number of clusters. This variant can be useful when the number of clusters in the data set is not available or cannot be approximated. However, the computational time required to find an optimum is much higher than for the other methods. Moreover, the value of the parameter σ influence the clustering results; this dependence represent a big disadvantage for the algorithm.

Some topics are mentioned for further study:

- Test the proposed algorithms with real data.
- Try other similarity measures, so we obtain a different updating rule.

- Interpret the γ probabilities from EM method as the likelihood of belonging to a cluster.
- Compare σ parameters for different methods, specially for “alternative” algorithm.

Appendix A: Matlab Codes

A.1 Batch k-means

```
function [w,winit,it,res] = batchkmeans(x,k,proto)
%batchkmeans.m looks for k clusters of a dataset x using the batch variant
%of the kmeans method.
[rn,cn] = size(x);
if nargin==2
    %Randomly choose prototypes w ( smart init -> w = rand(x) )
    r=randperm(rn,k);
    w=zeros(k,cn);
    wold=w;
    for i=1:k
        w(i,:)=x(r(i),:);
    end
end
if nargin==3
    w=proto;
    wold=zeros(k,cn);
end
winit=w;
%Loop -> Looking for centroids (minimizing cost function):
it=0;
while norm(w-wold) > 10^-5
    it=1+it;
    %Calculate winner (Index_i (x_j) for all i,j)
    Q=zeros(k,1); %Initialize Q
    lab=zeros(rn,1);
    for l=1:rn
        for j = 1:k
            Q(j,1) = norm(x(l,:) - w(j,:)); %Distance
        end
        [~,s] = min(Q);  %Winner...
        lab(l,1)=s;
    end
    %Compute w_i = (sum_j I_i (x_j) *x_j) / (sum_j I_j (x_j)) -> (N_i)
    wold=w;
    for j=1:k
        count=0;
```

```
        w(j,:)=0;
        for l=1:rn
            if lab(l,1) == j
                w(j,:)=x(l,:)+w(j,:);
                count=1+count;
            end
        end
        w(j,:)=w(j,:)/count;
    end
    res(it)=norm(w-wold);
%end loop
end
```

A.2 Batch Neural Gas

```
function [w,winit,it,res] = batchng(X, N)
%batchng.m looks for N clusters of a dataset X using the batch variant
%of the Neural Gas method.

    %% Data size
    [nr, nc] = size(X);

    %% Parameters
    lambdai = 1;
    lambdaf = 0.0001;

    %% Initialization
    w = zeros(N, nc);
    wold=w;
    r=randperm(nr,N);
    mind = min(X);
    maxd = max(X);
    for i = 1:N
        w(i,:)=X(r(i),:);
        %w(i,:) = unifrnd(mind, maxd);
    end
    winit=w;
    it=0;

    %% Main Loop
    while norm(wold-w)>10^-5
        %Update measures
```

```
        K=zeros(nr,N);
        for i=1:nr
            Q=zeros(N,1);
            for j=1:N
                Q(j) = norm(X(i,:)-w(j,:));
            end
            %Ranking
            [~,I]=sort(Q);
            K(i,:)=I;
        end
        wold=w;
        lambda = lambdai*(lambdaf/lambdai)^(it/100);
        it=it+1;
        w(:,:)=0;
        s=zeros(1,N);
        for i=1:nr
            for j=1:N
                h=exp(-(j-1)/lambda);
                s(K(i,j))=s(K(i,j)) + h ;
                w(K(i,j),:) = w(K(i,j),:) + (h * X(i,:));
            end
        end
        for j=1:N
            w(j,:)=w(j,:)/s(j);
        end
        res(it)=norm(wold-w);
        if it==1000
            break
        end
    end
end
```

A.3 Proposed k-means

```
function [w,winit,it,res] = proposedkmeans(x,k,proto)
%proposedkmeans.m looks for k clusters of a dataset x using gaussian
%function as similarity measure and an EM motivated updating rule .

%Randomly choose prototypes w ( smart init -> w = rand(x) )
[rn,cn] = size(x);

if nargin==2
```

```
    r=randperm(rn,k);
    w=zeros(k,cn);
    wold=w;
    for i=1:k
        w(i,:)=x(r(i),:);
        %w(i,:) = unifrnd(Xmin, Xmax);
    end
end
if nargin==3
    wold=zeros(k,cn);
    w=proto;
end

%sigma=100000; %CI
%sigma=0.5; %Mouse
%sigma=0.05; %Clusterdemo
%sigma=0.001;

winit=w;
%Loop -> Looking for centroids (minimizing cost function):
it=0;
while norm(w-wold) > 10^-5
    it=1+it;
    wold=w;
    Q=zeros(k,1); %Initialize Q
    lab=zeros(rn,1);
    for l=1:rn
        for j = 1:k
            Q(j,1) = exp(-norm(x(l,:) - w(j,:))^2)/(2*sigma); %Similarities
        end
        [~,s] = max(Q);  %Winner...
        lab(l,1)=s;
    end

    gamma=zeros(rn,k);
    for i=1:rn
        for j=1:k
            if j == lab(i)
                gamma(i,j)=exp(-(norm(x(i,:)-w(j,:))^2)/(2*sigma));
            end
        end
    end
    total = sum(sum(gamma));
```

```
    gamma=gamma/total;
    for j=1:k
        w(j,:)=0;
        numerator=zeros(1,cn);
        for l=1:rn
            numerator(1,:)=gamma(l,j)*x(l,:) + numerator;
        end
        w(j,:)=numerator/sum(gamma(:,j));
    end
    res(it)=norm(w-wold);
%end loop
end
```

A.4 Proposed Neural Gas

```
function [w,winit,it,res]=proposedng(x,k)
%proposedng.m finds k clusters in data x using a variant of the Neural Gas algort
%using a different updating rule derived by an EM approach using the gaussian
%similarity measure.

[nr,nc]=size(x);
w = zeros(k, nc);
wold=w;
r=randperm(nr,k);
for i = 1:k
    w(i,:)=x(r(i),:);
    %w(i,:) = unifrnd(mind, maxd);
end
winit=w;

%Parameters
%sigma=1000; %All
%sigma=10000;%CI
%sigma=10000000;%density
%sigma=100; %Random and cluster
lambdai=1;
lambdaf=0.01;
it=0;

%Main loop
while norm(wold-w)>10^-5
    %Update measures
```

```
    gamma=zeros(nr,k);
    lambda = lambdai*(lambdaf/lambdai)^(it/100);
    it=it+1;
    for i=1:nr
        Q=zeros(k,1);
        for j=1:k
            Q(j) = exp(-((norm(x(i,:)-w(j,:))^2)/(2*sigma)));
        end
        %Ranking
        [~,I]=sort(Q,'descend');
        %Gammas
        for j=1:k
            gamma(i,I(j)) = Q(j) * exp(-(((j)-1)/lambda));
        end
    end
    gamma=gamma/sum(sum(gamma));

    %Update prototypes
    wold=w;
    w(:,:)=0;
    for j=1:k
        for i=1:nr
            w(j,:)= gamma(i,j)*x(i,:) + w(j,:);
        end
        w(j,:)=w(j,:)/sum(gamma(:,j));
    end
    res(it)=norm(wold-w);
    %scatter(x(:,1),x(:,2))
    %hold on
    %scatter(w(:,1),w(:,2),'r','fill')
    %hold on
    %scatter(winit(:,1),winit(:,2),'g')
    %pause
    if it==1000
        break
    end
end
```

A.5 Alternative method

```
function w=findclustgrid(x)
%findclustgrid.m clusters the data set x in according to similarity measures
```

```
%using the RBF kernel.
%The number of clusters depends on the choice of the sigma parameter:
%For big sigma, the center of dataset is computed
%For an "small enough" sigma, the center of different clusters are found.
%For extremely small sigma, the prototypes hardly move from initial
%position

[nr,nc]=size(x);

if nc==3
    mny=floor(min(x(:,2))); mxy=ceil(max(x(:,2)));
    mnx=floor(min(x(:,1))); mxx=ceil(max(x(:,1)));
    mnz=floor(min(x(:,3))); mxz=ceil(max(x(:,3)));
    xaxis=mnx:(mxx-mnx)/5:mxx;
    yaxis=mny:(mxy-mny)/5:mxy;
    zaxis=mnz:(mxz-mnz)/5:mxz;
    w=[mean(x(:,1)),mean(x(:,2)),mean(x(:,3))];
    for i=1:length(xaxis)
        for j=1:length(yaxis)
            for l=1:length(zaxis)
                w=[w(:,1) , w(:,2), w(:,3); xaxis(i), yaxis(j), zaxis(l)];
            end
        end
    end
elseif nc==2
    mny=floor(min(x(:,2))); mxy=ceil(max(x(:,2)));
    mnx=floor(min(x(:,1))); mxx=ceil(max(x(:,1)));
    xaxis=mnx:(mxx-mnx)/5:mxx;
    yaxis=mny:(mxy-mny)/5:mxy;
    w=[mean(x(:,1)),mean(x(:,2))];
    for i=1:length(xaxis)
        for j=1:length(yaxis)
            w=[w(:,1) , w(:,2); xaxis(i), yaxis(j)];
        end
    end
else
    disp('No possible to cluster more than 3d data')
    return
end
k=size(w,1);
wold = zeros(k, nc);
winit=w;
```

```
%Parameters
%sigma = 0.002; normalized
%sigma=1; %Random data
%sigma=0.01; %clusterdata
%sigma=0.15; %Mouse data
%sigma=100; %CI 2 clusters
%sigma=50; %CI 4 clusters
sigma=10000;
it=0;

%Main loop
while norm(wold-w)>10^-5
    it=it+1;

    %Update prototypes
    wold=w;
    wnew = zeros(k, nc);
    for j=1:k
        s=0;
        for i=1:nr
            gamma = exp(-((norm(x(i,:)-w(j,:))^2)/(2*sigma)));
            s=gamma+s;
            wnew(j,:)= gamma*x(i,:) + wnew(j,:);
        end
        w(j,:)=wnew(j,:)/s;
    end
    res(it)=norm(wold-w);
    if it==2000
        break
    end
end
```

Bibliography

[1] D. Nebel, B. Hammer, K. Frohberg, T. Villmann. 2015. Median variants of learning vector quantization foor learning dissimilarity data. *Neurocomputing* Vol. 169, pp. 295-305

[2] M. Cotrell, B. Hammer, A. Hasenfuss, T. Villmann. 2006. Batch and Median Neural Gas. *Neural Networks* Vol. 19, No. 6, pp. 762-771

[3] J.B. MacQueen. 1967. Some Methods for classification and Analysis of Multivariate Observations. *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability - University of California Press* pp. 281-297

[4] S. Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans. Inform. Theory* Vol. IT-28, No. 2, pp. 129-137 (Originally a 1957 Bell Labs memorandum)

[5] L. Bottou, Y. Bengio. 1995. Convergence Properties of the K-Means Algorithms. *Advances in Neural Information Processing Systems* Vol. 7, pp. 585-592

[6] T. Martinetz, K. Schulten. 1991. A neural gas network learns topologies. *Artificial Neural Networks - Elsevier* pp. 397-402

[7] A.P. Dempster, N.M. Laird, D.B. Rubin. 1977. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society* Series B. Vol. 39, No. 1, pp. 1-38

[8] L.A. Garcia-Escudero, A. Gordaliza. 1999. Robustness Properties of k Means and Trimmed k Means. *Journal of the American Statistical Association* Vol. 94, No. 447, pp. 956-969

[9] L.V. Bijuraj. 2013. Clustering and its Applications . *Proceedings of National Conference on New Horizons in IT - NCNHIT* Vol. 1, pp. 169-172

[10] Ch. N. Kumar, K. Nageswara, A. Govardhan, N. Sandhya. 2015. Subset K-Means Approach for Handling Imbalanced-Distributed Data. *Emerging ICT for Bridging the Future - Proceedings of the 49th Annual Convention of the Computer Society of India CSI* Vol. 2, pp. 497-508

[11] F. Cai, Nhien-An Le-Khac, M.T. Kechadi. 2013. Clustering Approaches for Financial Data Analysis. *Proceedings of the 8th International Conference on Data Mining.* Vol. 1, pp. 105-111

[12] C. Saavedra, S. Moreno, R. Salas, H. Allende Robustness. 2006. Analysis of the Neural Gas Learning Algorithm. *Progress in Pattern Recognition, Image Analysis and Applications* Series 0302-9743, Vol. 4225, pp 559-568

[13] S. Kullback, R. Leibler. 1951. On information and sufficiency. *Annals of mathematical statistics* Vol. 22, pp. 79-86

[14] G. McLachlan. T. Krishnan. 1997. The EM Algorithm and Extensions. *John Wiley & Sons* New York, ISBN-13: 978-0471201700.

[15] T. Geweniger, F.M. Schleif, A. Hasenfuss, B. Hammer, T. Villmann. 2008. Comparison of cluster algorithms for the analysis of text data using Kolmogorov complexity. *Advances in Neuro-Information Processing* pp. 61-69

[16] J.P. Vert, K. Tsuda, B. Schölkopf. 2004. A primer on kernel methods. *Kernel Methods in Computational Biology* pp. 35-70

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, 25. 11 2016